プログラミング基礎

第6回
オブジェクト指向の概念
（クラス復習，補足）

前回

- 前回はクラスの作り方
  ⇒自分で必要になった型（クラス）を作る方法

  を学習

クラスの作り方

- これまでのまとめ

① クラスの名前は何か

② これから作ろうとしているクラスに必要な属性は何か
例：2次元ベクトルであればX，Y
名簿1件分のデータであれば名前，生年月日，住所，電話番号etc.

③ 各属性の値を設定する，取得する（返す）メソッドを作る
例：2次元ベクトルであればX，Yの値を設定，また取得するメソッド
名簿1件分のデータであれば名前などを設定，また取得するメソッド

④ 属性の値を基に，必要となった各種機能を実現するメソッドをクラスに追加する
例：2次元ベクトルであれば，X，Yを基に長さを求めるメソッドetc.
名簿1件分のデータであれば，年齢を返すメソッドなど

クラスの作り方

例えば2次元ベクトルのクラス

①

```
class Vector2D
{
    double x;
    double y;

    void SetValue(double newx, double newy)
    {
        x = newx;
        y = newy;
    }

    double GetLength()
    {
        double t = x * x + y * y;
        double l = Math.sqrt(t);
        return t;
    }
}
```

②

③

④

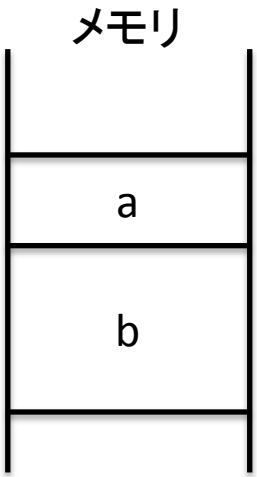
クラスの使い方

```
public class VectorLength
{
   static public void main(String[] args)
   {
      Vector2D v;
      v = new Vector2D();

      v.SetValue(1.0, 2.0);

     double l;
      l = v.GetLength();
     System.out.println("|v|=" + l);
   }
}
```

インスタンス（前回）

intやdoubleは以下のように記述するだけで自動的にメモリ上に値を記憶する領域ができる（確保される）

例：
int a;
double b;



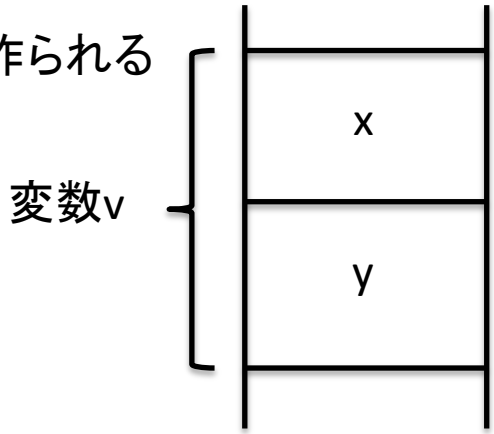
クラスは同じように書いても値を記憶する領域はできない

例：
Vector2D v0;

以下のように記述してはじめて**値を記憶する領域**が作られる

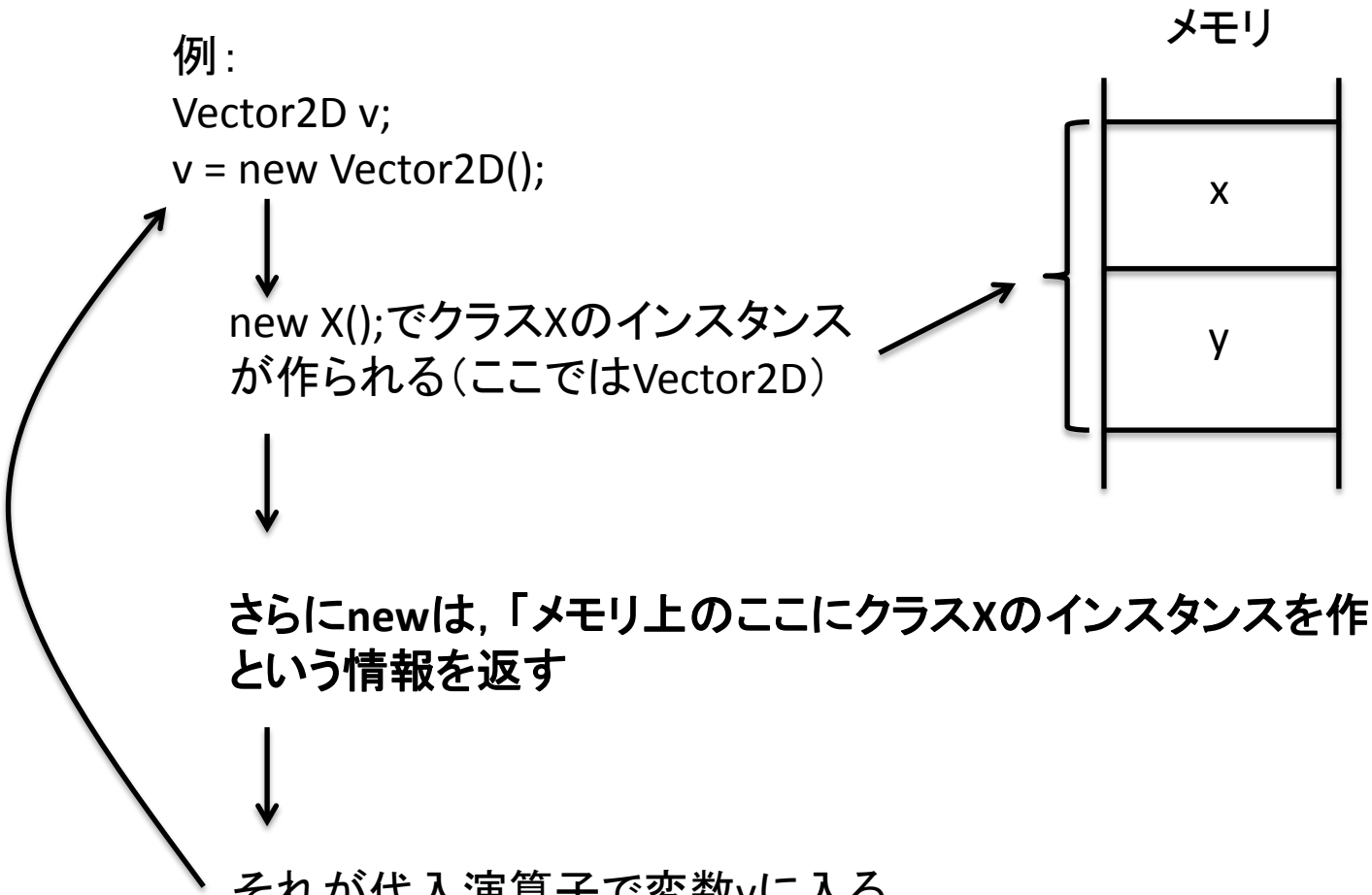
（⇒これを**インスタンス**と呼ぶ）

例：
Vector2D v;
v = new Vector2D();



newの役割（前回）

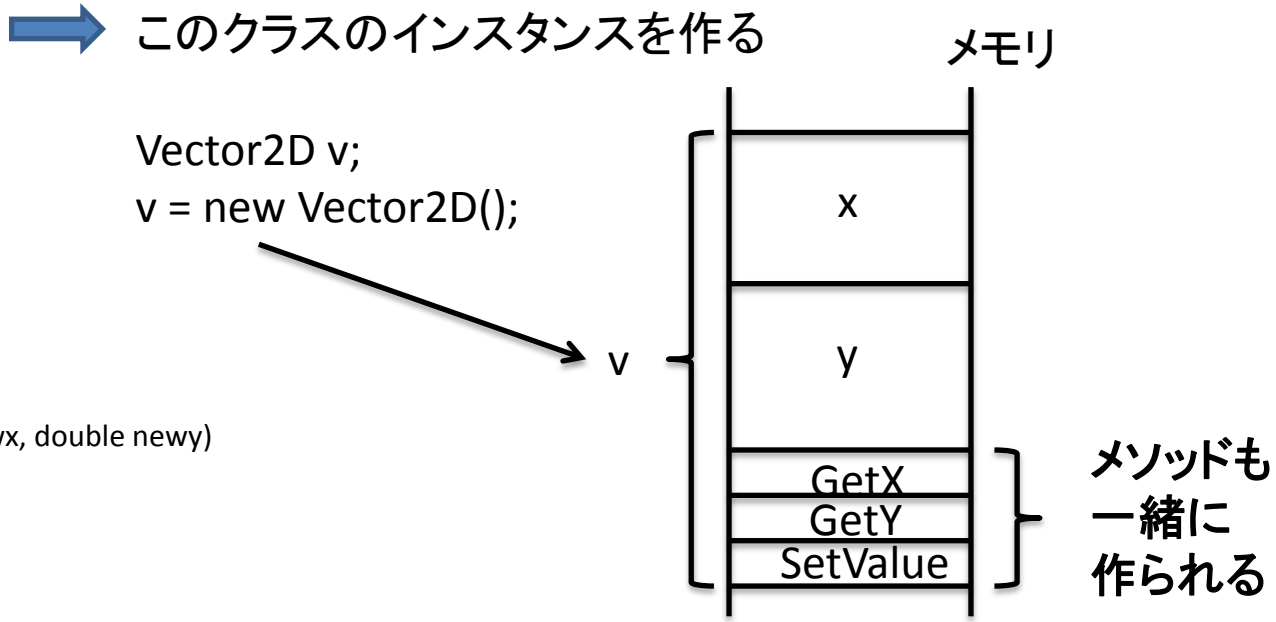
例：
Vector2D v;
v = new Vector2D();



インスタンスの補足

- クラスにメソッドが存在する場合

```
class Vector2D
{
   double x;
   double y;

   double GetX()
  {
     return x;
  }

  double GetY()
   {
      return y;
   }

   void SetValue(double newx, double newy)
   {
      x = newx;
      y = newy;
   }
}
```



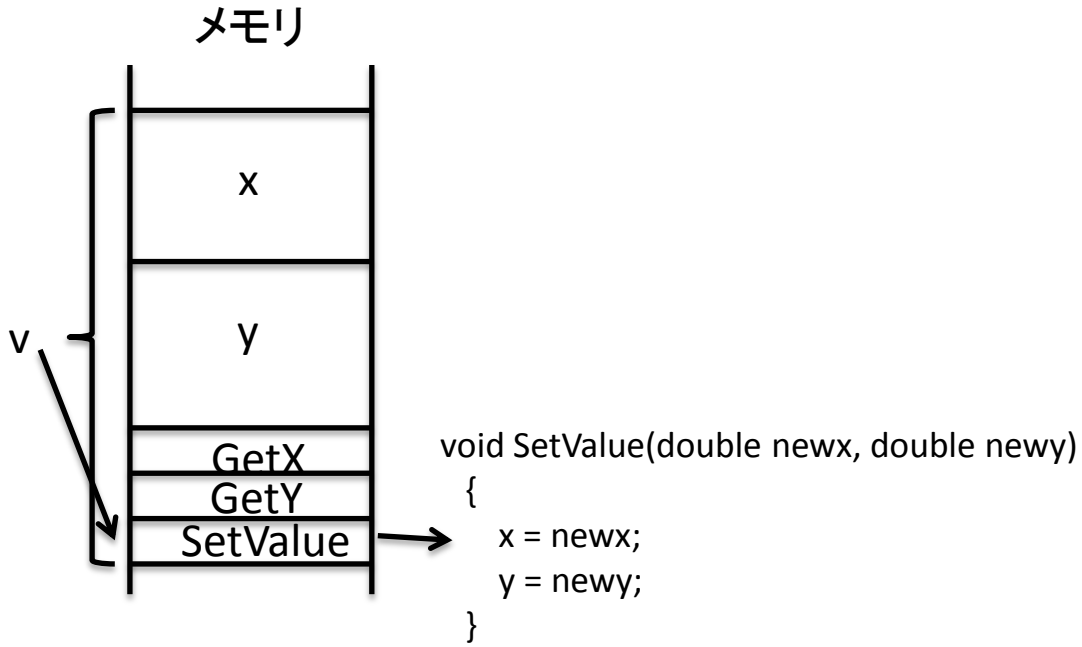
インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);
```

「vのSetValueメソッドを使います」



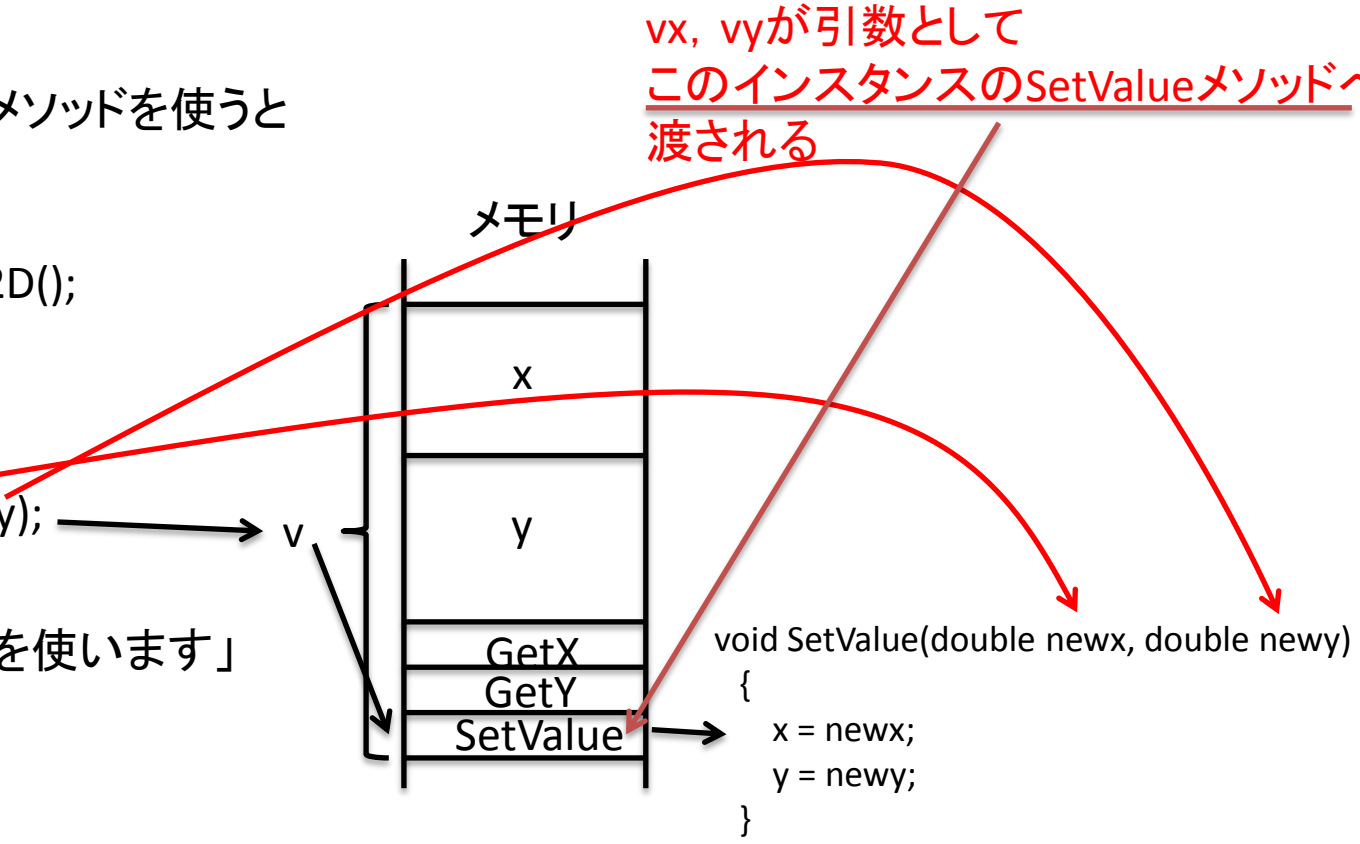
インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);
```

「vのSetValueメソッドを使います」



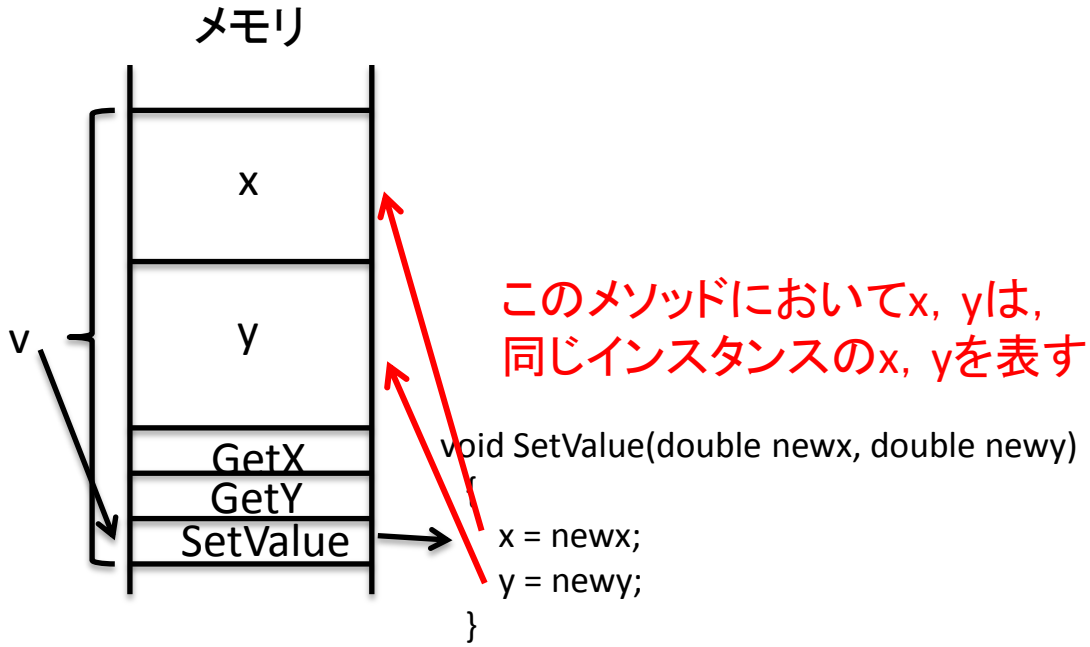
インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);
```

「vのSetValueメソッドを使います」



インスタンスの補足

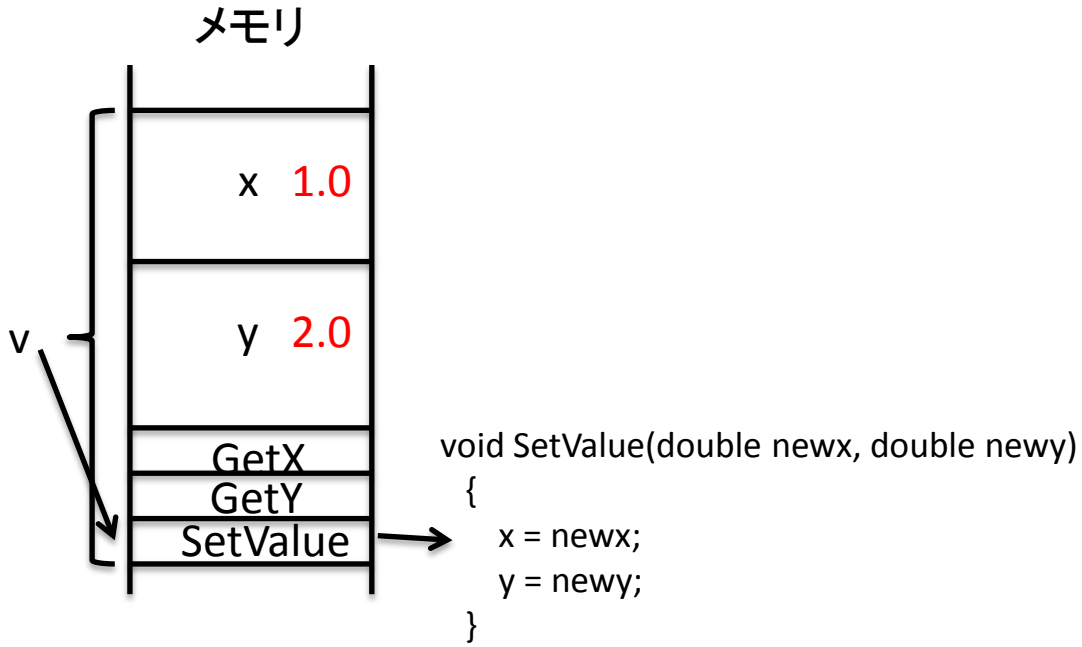
- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);
```

「vのSetValueメソッドを使います」

vのxは1.0，yは2.0になった

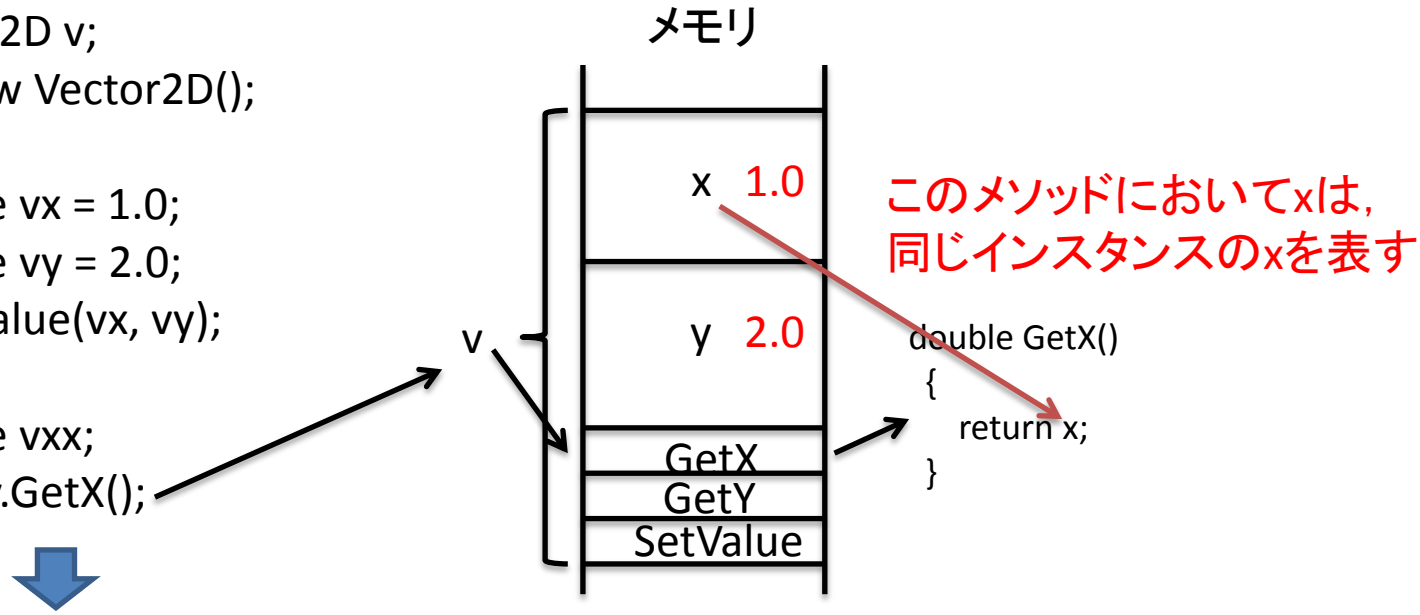


インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);

double vxx;
vxx = v.GetX();
```



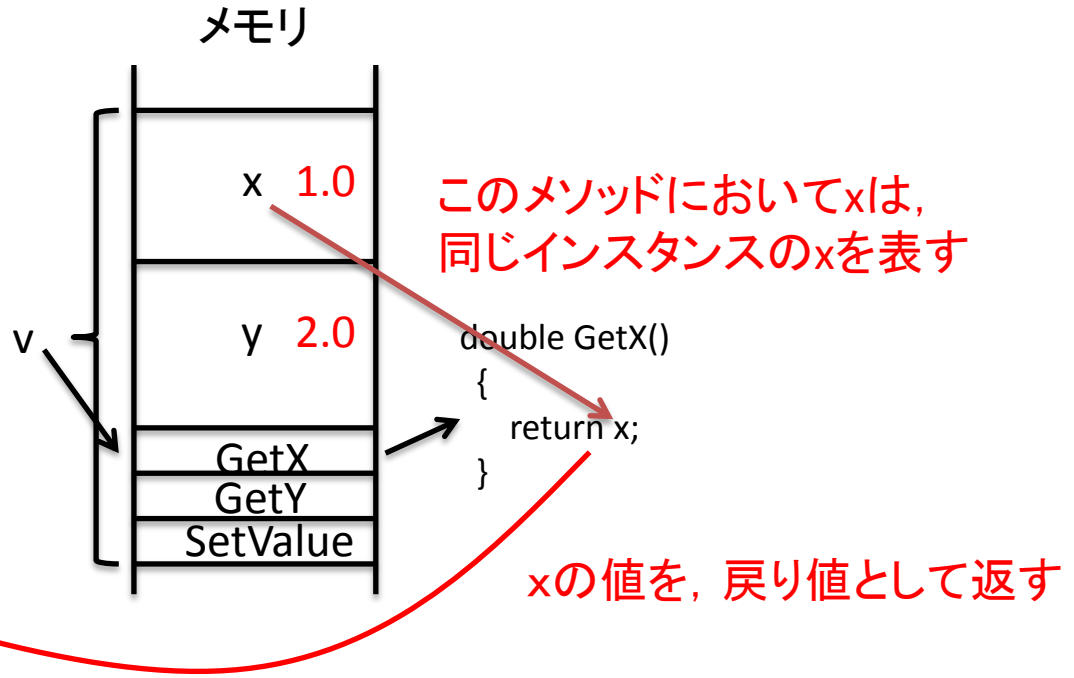
「vのGetXメソッドを使います」

インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);

double vxx;
vxx = v.GetX();
```



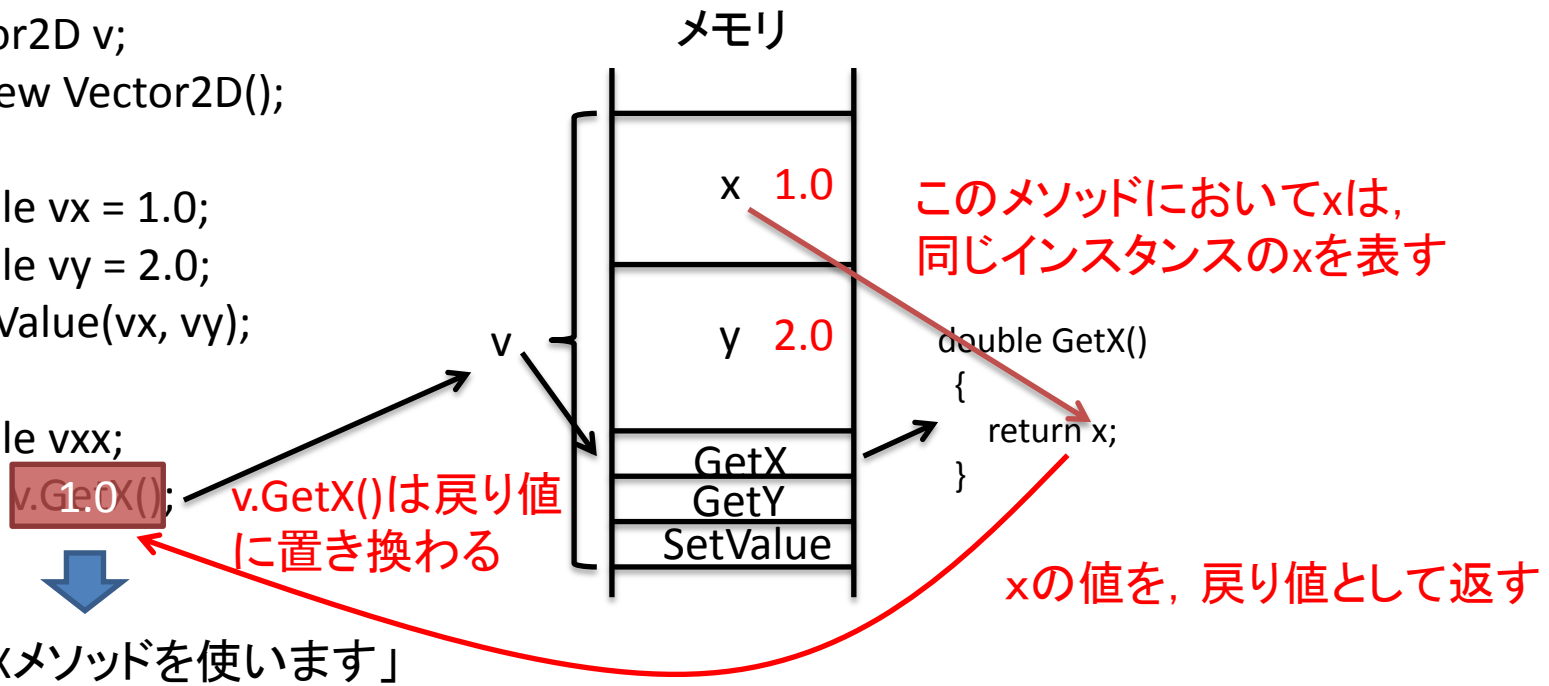
「vのGetXメソッドを使います」

インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);

double vxx;
vxx = v.GetX();
```



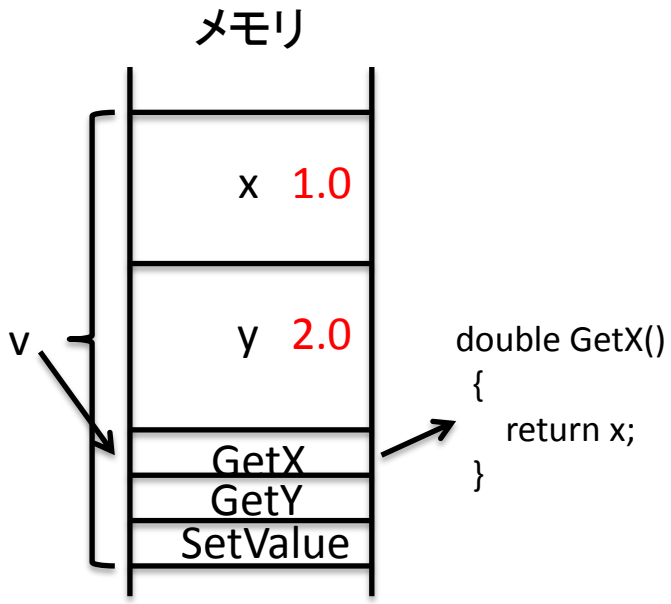
「vのGetXメソッドを使います」

インスタンスの補足

- クラスにメソッドが存在する場合

インスタンスのメソッドを使うと

```
Vector2D v;
v = new Vector2D();

double vx = 1.0;
double vy = 2.0;
v.SetValue(vx, vy);

double vxx;
vxx = v.GetX();
```



「vのGetXメソッドを使います」 変数vxxの値は1.0になる

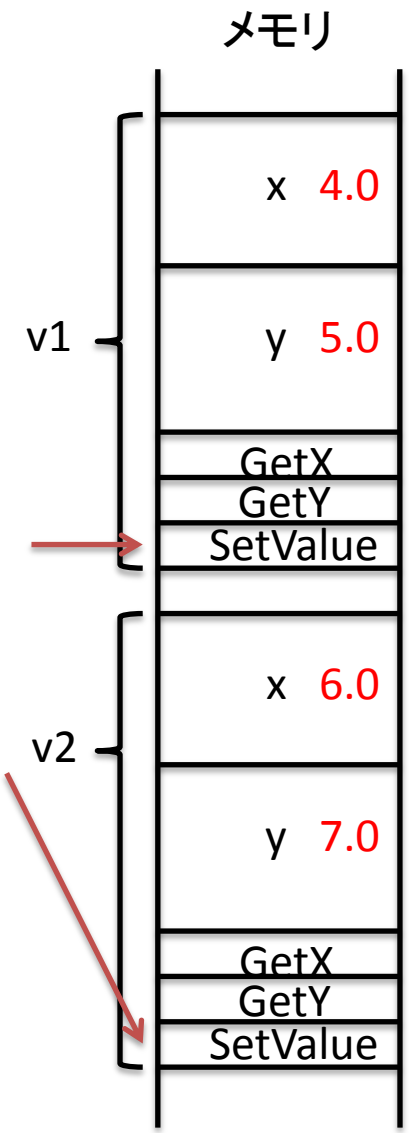
インスタンスの補足

- 複数のインスタンスを作成した場合も同様

```
Vector2D v1, v2;
v1 = new Vector2D();

double vx = 4.0;
double vy = 5.0;
v1.SetValue(vx, vy);

vx = 6.0;
vy = 7.0;
v2.SetValue(vx, vy);
```

「v1のSetValueメソッドを使います」

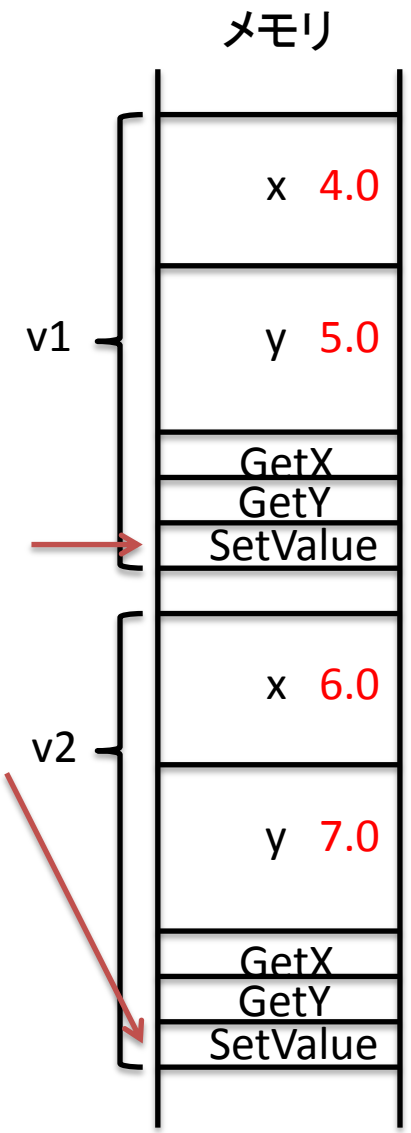
「v2のSetValueメソッドを使います」



注意

- 解説の便宜上⇒
  のような図を用いたが

  実際のJava内部では
  もっと処理効率の良い
  仕組みになっているので
  注意



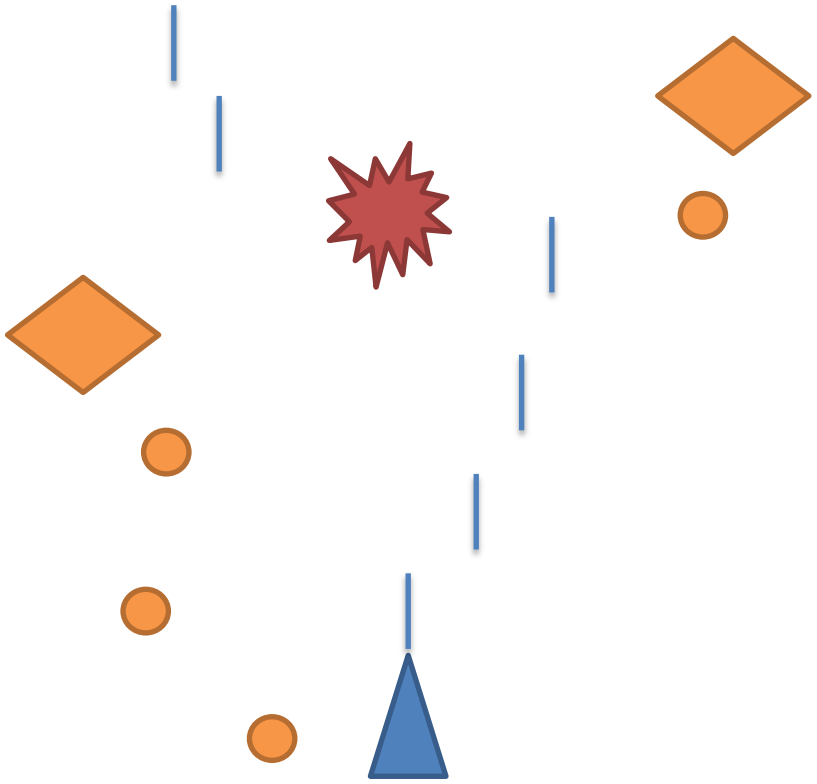
補足

- 前回の演習で,
  「円オブジェクトを生成する ;」
  という表現があったが,
  「オブジェクト」はインスタンスと同義なので,
  「円インスタンを生成する ;」
  ということ

- インスタンスを作るだけなのであれば,
  new Circle();
  ということになるが, **これではメモリのどこにインスタンスが作られたか分からないので,**
  適当な名前の変数を作り,
  Circle c;
  c = new Circle();
  としなければならないことに注意

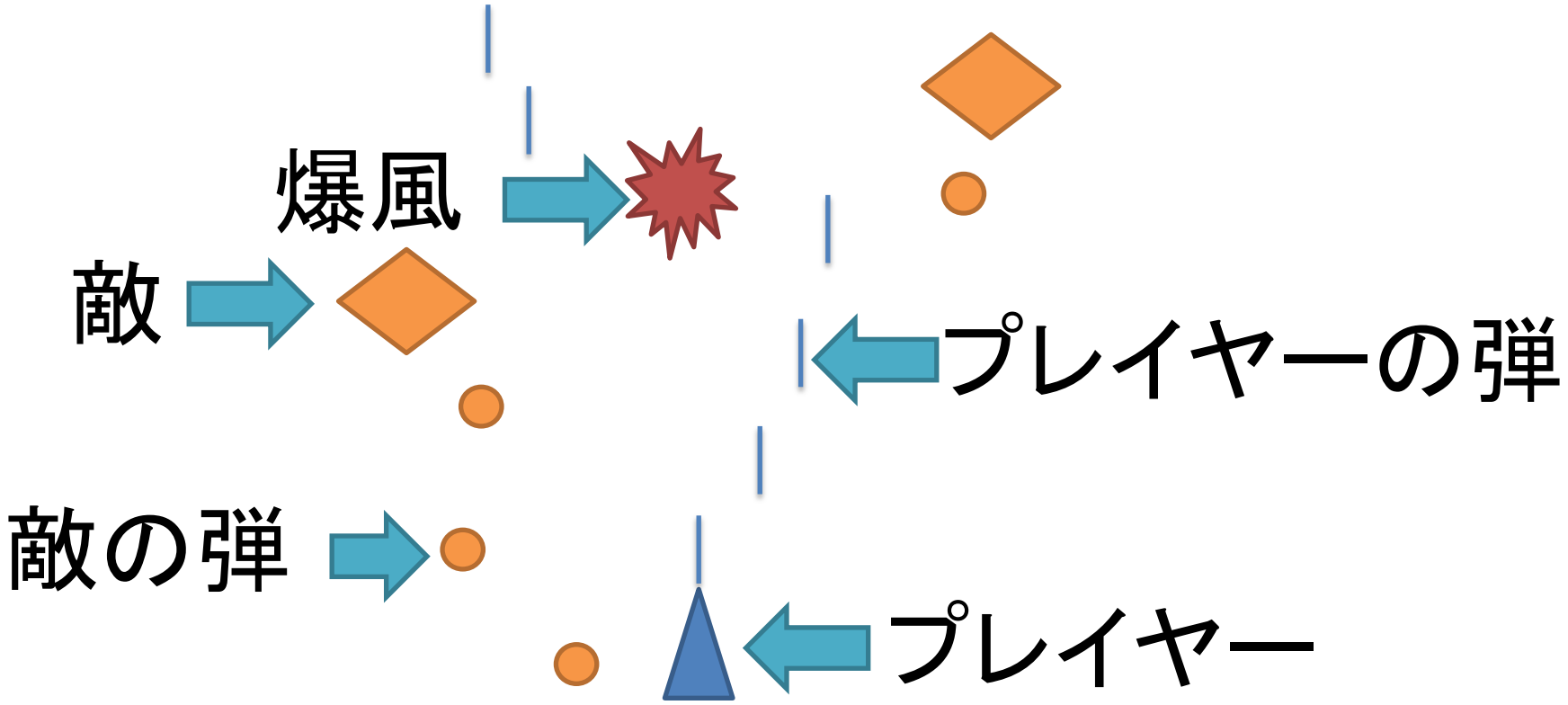
なぜクラスを使うのか

試しに以下のようなゲームを作ってみることを考える



なぜクラスを使うのか

試しに以下のようなゲームを作ってみることを考える



なぜクラスを使うのか

- ではクラスがなかったら…

```
static void main()
{
    int playerX;
    int playerY;
    int plyerLife;
    int enemyX = new int[10];
    int enemyY = new int[10];
    int enemyLife = new int[10];
    int playerBulletX = new int[10];
    int playerBulletY = new int[10];
    int playerBulletLife = new int[10];
    int enemyBulletX = new int[10];
    int enemyBulletY = new int[10];
    int enemyBulletLife = new int[10];
    int enemyBulletVx = new int[10];
    int enemyBulletVy = new int[10];
```

プレイヤーの位置

プレイヤーの体力

敵の位置

敵の体力

プレイヤーの弾の位置

プレイヤー弾の体力

敵の弾の位置

敵の弾が存在しているか

敵の弾の進行方向

スライドに入りきらないが，
他に，爆風用の位置などなど，同じようにすべて作っていくことになる

なぜクラスを使うのか

クラスを使うと…

```
static void main()
{
  int playerX;
  int playerY;
  int plyerLife;
  int enemyX = new int[10];
  int enemyY = new int[10];
  int enemyLife = new int[10];
  int playerBulletX = new int[10];
  int playerBulletY = new int[10];
  int playerBulletLife = new int[10];
  int enemyBulletX = new int[10];
  int enemyBulletY = new int[10];
  int enemyBulletLife = new int[10];
  int enemyBulletVx = new int[10];
  int enemyBulletVy = new int[10];
```

2次元での位置，向きを表す
⇒2次元ベクトルとしてまとめてしまえばよい

Vector2Dクラス（型）導入

```
class Vector2D
{
    int x;
    int y;
}
```

なぜクラスを使うのか

Vector2Dクラス（型）導入

```
static void main()
{
    Vector2D playerPosition;
    int plyerLife;
    Vector2D enemyPositions = new Vector2D[10];
    int enemyLife = new int[10];
    Vector2D playerBulletX = new Vector2D[10];
    int playerBulletLife = new int[10];
    Vector2D enemyBulletPositions = new Vector2D[10];
    int enemyBulletLife = new int[10];
    Vector2D enemyBulletVelocity = new Vector2D[10];
    Vector2D explosivePositions = new Vector2D[10];
    int explosiveLife = new int[10];
```

なぜクラスを使うのか

さらに見直せば,

```
static void main()
{
    Vector2D playerPosition;
    int plyerLife;
    Vector2D enemyPositions = new Vector2D[10];
    int enemyLife = new int[10];
    Vector2D playerBulletX = new Vector2D[10];
    int playerBulletLife = new int[10];
    Vector2D enemyBulletPositions = new Vector2D[10];
    int enemyBulletLife = new int[10];
    Vector2D enemyBulletVelocity = new Vector2D[10];
    Vector2D explosivePositions = new Vector2D[10];
    int explosiveLife = new int[10];
```

プレイヤーの情報

敵の情報

プレイヤーの弾の情報

敵の弾の情報

爆風の情報

さらにそれぞれをクラス化

なぜクラスを使うのか

```
static void main()
{
   Player player;
   Enemy enemies = new Enemy[10];
   PlayerBullet playerBullets = new PlayerBullets[10];
   EnemyBullets enemyBullets = new EnemyBullets[10];
   Explosive explosives = new Explosive[10];
```

```
class Player
{
   Vector2D position;
   int life;
}
```

```
class Enemy
{
   Vector2D position;
   Vector2D velocity;
   int life;
}
```

```
class Explosive
{
   Vector2D position;
   int life;
}
```

```
class PlayerBullet
{
   Vector2D position;
   int life;
}
```

```
class EnemyBullet
{
   Vector2D position;
   int life;
}
```

```
class Vector2D
{
   int x;
   int y;
}
```

どっちが分かりやすい？

```
static void main()
{
    int playerX;
    int playerY;
    int plyerLife;
    int enemyX = new int[10];
    int enemyY = new int[10];
    int enemyLife = new int[10];
    int playerBulletX = new int[10];
    int playerBulletY = new int[10];
    int playerBulletLife = new int[10];
    int enemyBulletX = new int[10];
    int enemyBulletY = new int[10];
    int enemyBulletLife = new int[10];
    int enemyBulletVx = new int[10];
    int enemyBulletVy = new int[10];
    int explosiveX = new int[10];
```

```
static void main()
{
    Player player;
    Enemy enemies = new Enemy[10];
    PlayerBullet playerBullets = new PlayerBullets[10];
    EnemyBullets enemyBullets = new EnemyBullets[10];
    Explosive explosives = new Explosive[10];
```

```
class Player
{
    Vector2D position;
    int life;
}

class Enemy
{
    Vector2D position;
    Vector2D velocity;
    int life;
}

class Explosive
{
    Vector2D position;
    int life;
}
```

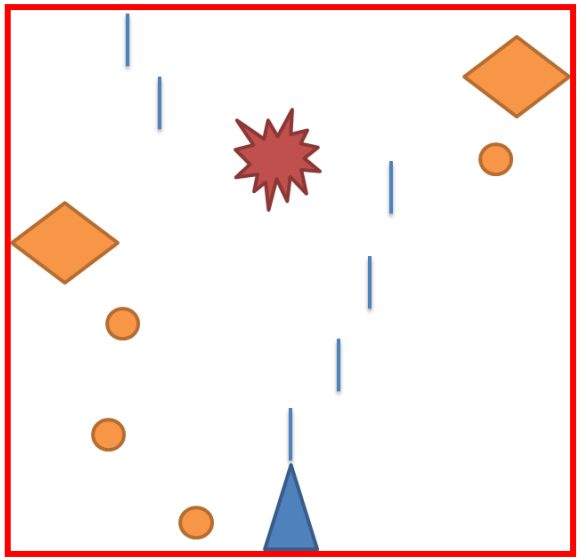
```
class PlayerBullet
{
    Vector2D position;
    int life;
}

class EnemyBullet
{
    Vector2D position;
    int life;
}

class Vector2D
{
    int x;
    int y;
}
```

オブジェクト指向

- ひとつの大きなシステムを
  オブジェクト（部品）ごとに分けて考え，
  それらが組み合わさって動くものとする．
  プログラムも，それに沿って記述するという
  考え方

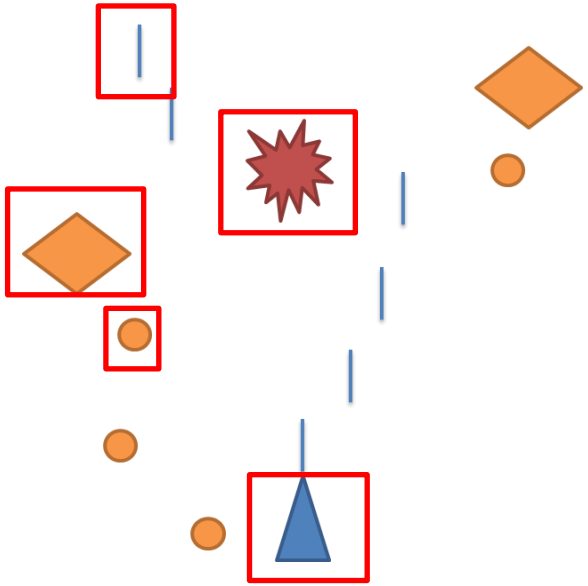
オブジェクト指向

- ひとつの大きなシステムを
  オブジェクト（部品）ごとに分けて考え，
  それらが組み合わさって動くものとする
  プログラムも，それに沿って記述するという
  考え方



オブジェクト指向

- ひとつの大きなシステムを
  オブジェクト（部品）ごとに分けて考え，
  それらが組み合わさって動くものとする．
  プログラムも，それに沿って記述するという
  考え方



クラスを作る時

- クラス名，必要な属性，機能（メソッド）が事細かに書かれた仕様書を渡されてそれを書くかもしれないし
- 「このような機能を持つクラスを作ってくれ」とだけ言われるかもしれないし
- すべて自分で書くプログラムならば，どのようなクラスを作れば効率的にプログラムが書けるか，自分で考える（⇒設計する）場合もある